Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 34 (2007) 1085–1106
computers & operations research
www.elsevier.com/locate/cor

A branch-and-cut algorithm for quadratic assignment problems based on linearizations

Güneş Erdoğan, Barbaros Tansel*

Department of Industrial Engineering, Bilkent University, Bilkent, Ankara, 06800, Turkey

Available online 31 August 2005

Abstract

The quadratic assignment problem (QAP) is one of the hardest combinatorial optimization problems known. Exact solution attempts proposed for instances of size larger than 15 have been generally unsuccessful even though successful implementations have been reported on some test problems from the QAPLIB up to size 36. In this study, we focus on the Koopmans–Beckmann formulation and exploit the structure of the flow and distance matrices based on a flow-based linearization technique that we propose. We present two new IP formulations based on the flow-based linearization technique that require fewer variables and yield stronger lower bounds than existing formulations. We strengthen the formulations with valid inequalities and report computational experience with a branch-and-cut algorithm. The proposed method performs quite well on QAPLIB instances for which certain metrics (indices) that we proposed that are related to the degree of difficulty of solving the problem are relatively high ($\geqslant 0.3$). Many of the well-known instances up to size 25 from the QAPLIB (e.g. nug24, chr25a) are in this class and solved in a matter of days on a single PC using the proposed algorithm.

Keywords: Quadratic assignment problem; Linearization; Branch-and-cut

1. Introduction

The Quadratic Assignment Problem (QAP) is the problem of assigning a set of n objects to another set of n objects so as to minimize the sum of the costs associated with pairs of assignments.

* Corresponding author.
E-mail addresses: egunes@bilkent.edu.tr (G. Erdoğan), barbaros@bilkent.edu.tr (B. Tansel).

0305-0548/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2005.05.027

QAP is NP-Hard [1] and among the most difficult NP-Hard problems. In fact, QAP is NP-Hard in the strong sense [1] meaning that finding an $\in$-approximate solution in polynomial time implies P = NP. It is also known that QAP is PLS-Complete [2], meaning that an exponential number of iterations is required in the worst case to find a local minimum for a certain type of neighborhood of solutions. Even though faster computers, specialized data structures, and algorithmic improvements have led to significant progress in solvable sizes of many NP-Hard problems (e.g. the Traveling Salesman, Vehicle Routing, Set Covering, Uncapacitated Facility Location, etc.), the QAP has been defiantly resisting all solution attempts beyond the size of $n > 15$ when the cost data is arbitrary. The largest solved instance of the QAP to date is of size 36 [3,4] while the largest solved size of, for example, the Traveling Salesman Problem has close to 25000 cities [5]. Most successful applications to date are limited to parallel implementations, which harness high amounts of computing power.

The computational status of the QAP poses a challenge: What new perspectives do we need to solve larger sizes of the QAP without having to rely on the high computing power of parallel processing? This paper takes up the challenge and presents a new modeling perspective on the QAP that leads to an effective branch-and-cut technique. The effectiveness of the proposed method is determined on the basis of six indices, which we also propose. These indices measure various features that are related to the degree of difficulty of solving the problem.

2. Problem definition

The problem is initially defined by Koopmans and Beckmann [6] in the context of assigning n facilities (machines) to n locations. Let f_{ij} be the annual flow between facilities i and j and d_{kl} be the distance between locations k and l. Each facility will be assigned to exactly one location, and each location gets assigned exactly one facility. Let $\boldsymbol{a} = (a(1), a(2), \ldots, a(n))$ be an *assignment* (a permutation of the integers $\{1, \ldots, n\}$) with $a(i)$ denoting the index of the location to which facility i is assigned. Define A to be the set of all such assignments. The QAP can be posed as

$$\min_{a \in A} \sum_{i,j} f_{ij} d_{a(i)a(j)}. \tag{1}$$

We refer to (1) as the *factorable* or Koopmans–Beckmann form of the QAP. In the general form, we take the cost of assigning facility i to location k and of facility j to location l as C_{ijkl} so that the problem becomes

$$\min_{a \in A} \sum_{i,j} C_{ia(i)ja(j)}. \tag{2}$$

If a nonzero cost $\hat{C}_{ik}$ is incurred for assigning facility i to location k, then the linear term $\sum_i \hat{C}_{ia(i)}$ is added to the objective function. One can re-define the costs C_{ijkl} (or $f_{ij}d_{kl}$) to be $C_{ijkl} + \hat{C}_{ik} + \hat{C}_{jl}$ to convert the problem to the pure quadratic form. Linear cost terms have been dropped in later formulations.

There are $n!$ assignments, which makes direct enumeration computationally prohibitive for $n > 14$. The general form requires the specification of n^4 cost elements, while the factorable form requires two $n \times n$ matrices $\boldsymbol{F} = [f_{ij}]$ and $\boldsymbol{D} = [d_{kl}]$, reducing the data requirement to $O(n^2)$.

Let x_{ik} be equal to 1 if facility i is assigned to location k, and 0 otherwise. An integer programming formulation of the factorable form is:

$$\min \quad \sum_{i,j,k,l=1}^{n} f_{ij} d_{kl} x_{ik} x_{jl} \tag{3}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{ik} = 1, \forall i = 1 \ldots n \tag{4}$$

$$\sum_{i=1}^{n} x_{ik} = 1, \forall k = 1 \ldots n \tag{5}$$

$$x_{ik} \in \{0, 1\}, \forall i, k = 1 \ldots n \tag{6}$$

Nonlinearity in the objective function is removed by linearizing the cost function. Even though many linearizations have been proposed for various special cases, two methods for linearizing the cost function have dominated the literature. In 1963, Lawler defined the variables $y_{ikjl} = x_{ik} x_{jl}$ so as to represent the *interaction* between two assignment decisions [7]. This interpretation was used by other authors to devise linearizations [8,9]. Later, Kaufmann and Broeckx defined the variables $w_{ik} = x_{ik} \sum_{j,l} f_{ij} d_{kl} x_{jl}$ to represent by w_{ik} the *contribution* of each assignment variable x_{ik} to the overall cost [10]. The linearization by Kaufmann and Broeckx is more compact in terms of the number of variables and constraints ($O(n^2)$), but the lower bounds generated by the corresponding relaxation of the IP are too weak to be of use. However, if Lawler's method is used, the number of variables increases to $O(n^4)$. In this case, lower bounds generated by the relaxation are much stronger, but the number of variables quickly exceeds computational reach as the problem size grows. Notably, high levels of degeneracy have been observed in Lawler's linearization [11]. Current integer programming models based on foregoing or other linearizations have not been able to cope with instances of size larger than 15 [12,11].

In the next section, we propose a new linearization based on a flow interpretation of the problem. Our branch-and-cut method, based on the proposed linearization, solves problems of size up to 30 in a CPU time of a few days on a single PC.

3. Flow-based linearized models

Note that the quadratic cost coefficients $f_{ij} d_{kl}$ represent the cost incurred when f_{ij} units of commodity are transported between locations that are d_{kl} units away. One may perceive the amount of flow between two facilities as a decision of its own. Even though these flow decisions may appear to be redundant, since they are determined as soon as the assignments are made, they are the real source of the transportation cost. To incorporate these decisions into a formulation, each facility may be perceived as a source of the commodity that it produces and sends over the network. The flow-based linear model is motivated by this observation and is a multicommodity flow formulation:

Let y_{ijk} be the amount of flow of commodity (facility) k from location i to location j.

(IP1)

$$\min \quad \sum_{i,j,k=1}^{n} d_{ij} y_{ijk}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} y_{ijk} = \left(\sum_{l=1}^{n} f_{kl} \right) x_{ki}, \forall i, k = 1 \ldots n \tag{7}$$

$$\sum_{i=1}^{n} y_{ijk} = \sum_{l=1}^{n} f_{kl} x_{lj}, \forall j, k = 1 \ldots n \tag{8}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1 \ldots n \tag{9}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j = 1 \ldots n \tag{10}$$

$$x_{ij} \in \{0, 1\}, \forall i, j = 1 \ldots n \tag{11}$$

$$y_{ijk} \geqslant 0, \forall i, j, k = 1 \ldots n \tag{12}$$

$$y_{ijk} \leqslant \max_{l=1 \ldots n} f_{kl}, \forall i, j, k = 1 \ldots n. \tag{13}$$

Theorem 1. *Let $\boldsymbol{x}$ be a feasible solution to an instance of the QAP defined by matrices $\boldsymbol{F}$* and *$\boldsymbol{D}$ with objective value $z_{\mathrm{QAP}}(\boldsymbol{x})$. Then, there exists a unique vector $\boldsymbol{y}$ such that $(\boldsymbol{x}, \boldsymbol{y})$ is feasible to IP1 with objective value $z_{\mathrm{IP1}}(\boldsymbol{x}, \boldsymbol{y}) = z_{\mathrm{QAP}}(\boldsymbol{x})$.*

Proof. With $\boldsymbol{x}$ being feasible to the QAP, constraints (9)–(11) of IP1 are satisfied. For each $k \in \{1, \ldots, n\}$, let $a(k)$ be the location index i for which $x_{ki} = 1$. Similarly, for each location j, let $a^{-1}(j)$ be the facility index l for which $x_{lj} = 1$. Since $x_{ki} = 0 \ \forall \ i \neq a(k)$, (7) and (12) imply that $y_{ijk} = 0 \ \forall \ i \neq a(k)$ and $k \in \{1, \ldots, n\}$. Consequently, the left side of (8) gives $y_{a(k)jk}$ (because all terms except for $i = a(k)$ are zero), while the right side of (8) gives $f_{ka^{-1}(j)}$ (because all terms except for $l = a^{-1}(j)$ are zero). Hence, $\boldsymbol{y}$ is uniquely determined by the equations

$$y_{a(k)jk} = f_{ka^{-1}(j)} \forall j, k \in \{1, \ldots, n\}$$

and

$$y_{ijk} = 0 \ \forall \ i \neq a(k) \text{ and } j, k \in \{1, \ldots, n\}.$$

Solution $\boldsymbol{y}$ constructed in this way satisfies (12) and (13). It also satisfies (8) by construction. The only remaining possibility to be checked is constraint (7). If $i \neq a(k)$, then (7) gives zero on both sides. If $i = a(k)$, the left side of (7) is $\sum_{j=1}^{n} y_{a(k)jk}$ while the right side is $F_k \equiv \sum_{l=1}^{n} f_{kl}$. Since $y_{a(k)jk} = f_{ka^{-1}(j)}$ by construction, the left side is $\sum_{j=1}^{n} f_{ka^{-1}(j)}$, which is the same as F_k. This proves the uniqueness and feasibility of $(\boldsymbol{x}, \boldsymbol{y})$ to IP1 for each feasible $\boldsymbol{x}$ to QAP.

To prove $z_{\mathrm{IP1}}(\boldsymbol{x},\boldsymbol{y}) = z_{\mathrm{QAP}}(\boldsymbol{x})$, observe that $f_{kl}d_{ij}x_{ki}x_{lj} = 0$, unless $k = a^{-1}(i)$ and $l = a^{-1}(j)$, in which case it is $f_{a^{-1}(i)a^{-1}(j)}d_{ij}$. Since the objective value of IP1 gives $\sum_{i,j=1}^{n} d_{ij} f_{a^{-1}(i)a^{-1}(j)}$, it is the same as $\sum_{i,j,k,l=1}^{n} f_{kl}d_{ij}x_{ki}x_{lj}$. □

In the foregoing formulation, assigning a facility (commodity) to a location may be interpreted as placing a supply of that commodity and demands for other commodities at that location. The formulation does not differentiate between the flows of commodities in terms of transportation costs. Based on this observation, a single commodity network flow model is constructed as follows.

Let y_{ij} be the amount of flow from location i to location j,

$$M_k = \max_{\substack{l,m\in[1,\ldots,n]\\ l\neq k}} f_{lm} - f_{km},$$

$$F_k^{out} = \sum_{l=1}^{n} f_{kl}, \text{ and}$$

$$F_k^{\mathrm{in}} = \sum_{l=1}^{n} f_{lk}.$$

(IP2)

$$\min \quad \sum_{i,j=1}^{n} d_{ij}y_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} y_{ij} = \sum_{k=1}^{n} F_k^{\mathrm{out}}x_{ki}, \forall i = 1\ldots n \tag{14}$$

$$\sum_{j=1}^{n} y_{ji} = \sum_{k=1}^{n} F_k^{\mathrm{in}}x_{ki}, \forall i = 1\ldots n \tag{15}$$

$$y_{ij} \leqslant \sum_{\substack{l=1\\ l\neq k}}^{n} f_{kl}x_{lj} + M_k(1 - x_{ki}), \forall i, j, k = 1\ldots n \tag{16}$$

$$y_{ij} \geqslant \min_{k,l=1\ldots n} f_{kl}, \forall i, j = 1\ldots n \tag{17}$$

$$y_{ij} \leqslant \max_{k,l=1\ldots n} f_{kl}, \forall i, j = 1\ldots n \tag{18}$$

and (9), (10), (11).

Theorem 2. *Let* $\boldsymbol{x}$ *be a feasible solution to an instance of the* QAP. *Then, there exists a unique* $\boldsymbol{y}$ *such that* $(\boldsymbol{x},\boldsymbol{y})$ *is feasible to* IP2 *with objective function value* $z_{\mathrm{IP2}}(x, y) = z_{\mathrm{QAP}}(x)$.

Proof. With $\boldsymbol{x}$ being feasible to the QAP, constraints (9)–(11) of IP2 are satisfied. With $a^{-1}(i)$ denoting the facility index k for which $x_{ki} = 1$, (14) and (15) give, respectively, that $\sum_{j=1}^{n} y_{ij} = \sum_{k=1}^{n} F_{a^{-1}(i)}^{\mathrm{out}}$ and $\sum_{j=1}^{n} y_{ji} = \sum_{k=1}^{n} F_{a^{-1}(i)}^{\mathrm{in}}$ for $i = 1, \ldots, n$. Constraint (16) sets the exact upper bound of each flow as

$y_{ij} \leqslant f_{a^{-1}(i)a^{-1}(j)}$. This upper bound must be satisfied as an equality, otherwise constraints (14) and (15) are violated. This proves the uniqueness and feasibility of $(\boldsymbol{x}, \boldsymbol{y})$ to IP2 for each feasible $\boldsymbol{x}$ to QAP.

To prove $z_{\mathrm{IP2}}(\boldsymbol{x}, \boldsymbol{y}) = z_{\mathrm{QAP}}(\boldsymbol{x})$, observe that $y_{ij} = f_{a^{-1}(i)a^{-1}(j)}$, implying that the objective value of IP2 is $\sum_{i,j=1}^{n} d_{ij} f_{a^{-1}(i)a^{-1}(j)}$ which is the same as $\sum_{i,j,k,l=1}^{n} f_{kl} d_{ij} x_{ki} x_{lj}$. □

IP1 has $O(n^3)$ variables and $O(n^2)$ constraints, whereas IP2 has $O(n^2)$ variables and $O(n^3)$ constraints. Both formulations are valid for arbitrary distance data.

In both formulations, the assignment variables $\boldsymbol{x}$ force the $\boldsymbol{y}$ variables to assume uniquely determined values from the flow matrix. Using a similar approach, in which the $\boldsymbol{x}$ values induce distance values on the variable set $\boldsymbol{y}$, conjugate formulations for which the roles of $\boldsymbol{D}$ and $\boldsymbol{F}$ are interchanged may be constructed. The conjugate formulations will be referred to as IP1′ and IP2′ in the rest of the paper.

4. Valid inequalities

In this section, we elaborate on IP2′ which is the single commodity formulation of the QAP with variables representing induced distances between facilities, and present some valid inequalities. To avoid confusion with the variables representing induced flows between locations, we write $\boldsymbol{t}$ instead of $\boldsymbol{y}$. That is, in obtaining IP2′ from IP2, we write t_{ij} in place of y_{ij} and use d_{kl} in place of f_{kl}. Note also that F_k^{out} and F_k^{in} are replaced by $D_k^{\mathrm{out}} = \sum_{l=1}^{n} d_{kl}$ and $D_k^{\mathrm{in}} = \sum_{l=1}^{n} d_{lk}$, respectively.

4.1. Triangle inequalities

We now restrict attention to distance matrices $\boldsymbol{D}$ that are symmetric and triangulated. That is, we assume $d_{ij} = d_{ji} \forall i, j = 1, \ldots, n$ and that $d_{ij} \leqslant d_{ik} + d_{kj}, \forall i, j, k$. In IP2′, the induced distances will be a permutation of the original distances. Consequently, the triangle inequalities are still valid when expressed in terms of the $\boldsymbol{t}$ variables representing the induced distances.

Theorem 3. *For an instance of the* QAP *whose distance matrix obeys the triangle inequalities, the inequalities* $t_{ij} \leqslant t_{ik} + t_{kj}, \forall i, j, k, i \neq j \neq k$ *are valid for* IP2′.

Proof. Let $(\boldsymbol{x}, \boldsymbol{t})$ be any feasible solution to IP2′ and consider an arbitrary triplet of distinct facility indices i, j, k. Let a, b, c be the location indices for which $x_{ia} = x_{jb} = x_{kc} = 1$. Then $t_{ij} = d_{ab}$, $t_{ik} = d_{ac}$, and $t_{kj} = d_{cb}$. Since the original distance matrix satisfies the triangle inequalities, we have $d_{ab} \leqslant d_{ac+} d_{cb}$, which implies that $t_{ij} \leqslant t_{ik+} t_{kj}$. □

Another interpretation of the triangle inequality is $\max_{\substack{i,j=1\ldots n \\ i \neq j}} d_{ij} - d_{ik} - d_{kj} \leqslant 0, \forall k = 1 \ldots n$. Let $T_k = \max_{\substack{i,j=1\ldots n \\ i \neq j}} d_{ij} - d_{ik} - d_{kj}, \forall k = 1 \ldots n$. A more general form of triangle inequalities is defined using T_k as follows.

Theorem 4. *The inequalities* $t_{ij} - t_{ik} - t_{kj} \leqslant \sum_{l=1}^{n} T_l x_{kl}, \forall i, j, k, i \neq j \neq k$ *are valid for* IP2′.

Proof. Let $(\boldsymbol{x}, \boldsymbol{t})$ be any feasible solution to IP2′, and consider an arbitrary triplet of distinct facility indices i, j, k. Let a, b, c be the location indices, for which $x_{ia} = x_{jb} = x_{kc} = 1$. Then $t_{ij} = d_{ab}$, $t_{ik} = d_{ac}$, and $t_{kj} = d_{cb}$. Since the original distance matrix satisfies the inequality $d_{ab} - d_{ac} - d_{cb} \leqslant T_c$ (by definition of T_c), we can conclude that $t_{ij} - t_{ik} - t_{kj} \leqslant \sum_{l=1}^{n} T_l x_{kl}$. □

Note that violated triangular inequalities can be identified in $O(n^3)$ time by simply checking all the inequalities. Violation of the generalized form of triangle inequalities can be identified by computing and storing $\sum_{l=1}^{n} T_l x_{kl}$ for each k in $O(n^2)$ time and then checking all the inequalities in a time bound of $O(n^3)$ and a space requirement of $O(n)$.

4.2. Upper bound inequalities

Let $I = \{1, \ldots, n\}$ and $I_i = I \backslash \{i\} \forall i \in I$. For $j \in \{1, \ldots, n-1\}$, define $TD_{ij} = \max_{\substack{L \subset I_i \\ |L| = j}} \sum_{l \in L} d_{il}$. With this definition, TD_{ij} gives the sum of the largest j distances from location i.

Theorem 5. *The inequalities* $\sum_{j \in J} t_{ij} \leqslant \sum_{k=1}^{n} TD_{k|J|} x_{ik}, \forall i, J$ *are valid for* IP2′, *where* $J \subseteq I_i$.

Proof. Assume the contrary. Then, there exists a feasible solution $(\boldsymbol{x}^*, \boldsymbol{t}^*)$ to IP2′, for which there exists a facility i and a set $J \subseteq I_i$, such that $\sum_{j \in J} t^*_{ij} > \sum_{k=1}^{n} TD_{k|J|} x^*_{ik}$. Let $x^*_{ik'} = 1$, and let L bethe set of locations assigned to the set of facilities J. Note that $|J| = |L|$. Then $\sum_{j \in J} t^*_{ij} = \sum_{l \in L} d_{k'l} \leqslant TD_{k'|J|}$, contradicting the assumed violation. □

Violated lower bound inequalities can be identified by sorting and storing t^*_{ij} values for all i ($O(n^2 \log n)$), computing and storing $\sum_{k=1}^{n} TD_{k|J|} x^*_{ik}$ for all i and $|J|(O(n^3))$, and comparing the sum of minimum $|J| t^*_{ij}$ values with $\sum_{k=1}^{n} TD_{k|J|} x^*_{ik}$ for all i and $|J|(O(n^2))$, at a total cost of $O(n^3)$ time and $O(n^2)$ space.

4.3. Constructed inequalities

The valid inequalities we have given so far have exploited certain properties of the distance matrix. We now switch attention to arbitrary distance (and flow) matrices. Suppose that we want to construct a valid inequality of the form

$$t_{ij} \leqslant \sum_{k=1}^{n} \alpha_k x_{ik} + \sum_{k=1}^{n} \beta_k x_{jk}, \forall i, j, \tag{19}$$

where α_k, β_k are constants to be determined.

Theorem 6. *If* α_k *and* β_k *obey the constraints* $\alpha_i + \beta_j \geqslant d_{ij}, \forall i, j : i \neq j$, *then the constraint set* $t_{ij} \leqslant \sum_{k=1}^{n} \alpha_k x_{ik} + \sum_{k=1}^{n} \beta_k x_{jk}, \forall i, j$ *is valid for* IP2′.

Proof. Assume the contrary. Then, α_k and β_k obey the constraints $\alpha_i + \beta_j \geqslant d_{ij}, \forall i, j : i \neq j$, but there exists a solution $(\boldsymbol{x}^*, \boldsymbol{t}^*)$ of IP2′ such that $t^*_{ij} > \sum_{k=1}^{n} \alpha_k x^*_{ik} + \sum_{k=1}^{n} \beta_k x^*_{jk}$ for some i and j, where $i \neq j$.

Let $x^*_{ik} = 1$ and $x^*_{jl} = 1$, where $k \neq l$. Then $t^*_{ij} = d_{kl} > \sum_{k=1}^{n} \alpha_k x^*_{ik} + \sum_{k=1}^{n} \beta_k x^*_{jk} = \alpha_k + \beta_l \geqslant d_{kl}$, which is a contradiction. □

For a given fractional solution $(\boldsymbol{x}^*, \boldsymbol{t}^*)$ to IP2′, a most violated valid inequality can be computed by solving the following linear program.

(LP1)

$$z^*_{\mathrm{LP1}} = \max t^*_{ij} - \sum_{k=1}^{n} \alpha_k x^*_{ik} - \sum_{k=1}^{n} \beta_k x^*_{jk}$$

$$\text{s.t. } \alpha_i + \beta_j \geqslant d_{ij}, \forall i, j : i \neq j.$$

All violated inequalities of the form (19) can be identified by solving LP1 $2\binom{n}{2}$ times, with varying objective function coefficients. A violated inequality is found if $z^*_{\mathrm{LP1}} > 0$. LP1 consists of $2n$ variables and $2\binom{n}{2}$ constraints, and provides an upper bound on the distance between two facilities. The idea may be generalized to impose bounds on the sum of distances between m facilities, where $2\binom{n}{m}$ instances of similar linear programs with mn variables and $m!\binom{n}{m}$ constraints must be solved. In our implementation, we have resorted to a heuristic way of identification to avoid solving exponentially many linear programs. Details of the heuristic are given below.

For $m = 2$, we solve a single instance of LP1 for each facility i, with $-x^*_{ik}$ as the cost coefficient for α_k, and $-(1 - x^*_{ik})/(n-1)$ as the cost coefficient for β_k. Cost coefficients of $\boldsymbol{\beta}$ describe an imaginary facility which is partially assigned to every possible location in a way that does not contradict facility i. The optimum solution $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ of this particular instance, in a sense, gives the best linearization for the current assignment of locations to facility i. We apply this valid inequality to facility pairs $(i, j)\ \forall j \neq i$. Valid inequalities constructed in this way require solving n instances of LP1 $\left(\text{instead of } m!\binom{n}{2}\right)$. Violated valid inequalities of this type can be identified by computing and storing $\sum_{k=1}^{n} \alpha_k x^*_{ik}$ for each i and $\sum_{k=1}^{n} \beta_k x^*_{jk}$ for $j \neq i$ $(O(n^2))$, and comparing t^*_{ij} with $\sum_{k=1}^{n} \alpha_k x^*_{ik} + \sum_{k=1}^{n} \beta_k x^*_{jk}$ $(O(n^3))$, resulting in a time bound of $O(n^3)$ and a space requirement of $O(n)$.

Note that, to decrease the computational burden of solving linear programs, one may assume equality of variables with the same subscript (i.e. $\alpha_k = \beta_k \forall k$), which decreases the number of variables by $1/m$ and the number of constraints by $1/(m!)$. The resulting valid inequalities constructed by solving these reduced linear programs are slightly weaker. For $m = 3$, we solve a single linear program with this assumption, and with all objective function coefficients being equal to $-1/n$. This way, we compute a single set of coefficients that give the best possible linearization for the case when assignment variables are equally divided. Inequalities constructed in this way can be identified by computing and storing $\sum_{k=1}^{n} \alpha_k x^*_{ik}$ for all i $(O(n^2))$, and comparing the sum of distances between every three facility (a, b, c) (i.e. $t^*_{ab} + t^*_{ba} + t^*_{ac} + t^*_{ca} + t^*_{bc} + t^*_{cb}$) with $\sum_{k=1}^{n} \alpha_k x^*_{ak} + \sum_{k=1}^{n} \alpha_k x^*_{bk} + \sum_{k=1}^{n} \alpha_k x^*_{cik}$ $(O(n^3))$, resulting in a time bound of $O(n^3)$ and a space requirement of $O(n)$. For $m > 3$, construction and identification processes become computationally prohibitive. Hence, we have disregarded valid inequalities corresponding to $m > 3$.

The valid inequalities presented in this section impose upper bounds on linear combinations of the decision variables. Variants of valid inequalities, in which the same combinations are bounded below, can be similarly constructed. As a final note, we note that, in terms of improving the optimum LP relaxation

value, the most effective one among the proposed valid inequalities we have presented is the one that uses the triangle inequalities.

5. Computational progress in solving the QAP

Before we present our computational results with the flow-based linearization, we give a historical sketch of the computational status of the QAP. A collection of instances and respective solutions, QAPLIB [13], is available online to benchmark efficiency of solution methods. Although many different classes of instances exist in the QAPLIB, the computational improvement for the QAP may best be explained by the progress in solving the notoriously difficult instances of Nugent et al. [14]. These are the most commonly used instances for testing computational efficiency. The original set consists of eight instances of sizes 5, 6, 7, 8, 12, 15, 20, and 30. Distance matrices for sizes 5 and 7 represent almost rectangular grid graphs. For sizes 6, 8, 12, 15, 20, and 30, the distance matrix represents grids of 2*3, 2*4, 3*4, 3*5, 4*5, and 5*6, respectively. Later, instances of sizes 14, 16, 17, 18, 21, 22, 24, and 25 were added to the original set by Clausen and Perregaard [15] by deleting certain rows and columns of flow and distance matrices of larger instances. Likewise, Anstreicher et al. [16] constructed instances of sizes 27 and 28 in the same way.

Nugent et al. [14] solved instances nug05, nug06, nug07, and nug08 to optimality using complete enumeration. Burkard and Stratmann [17] solved nug12 in 29.325 s on a CDC-CYBER76 machine and Burkard and Derigs [18] solved nug15 using a branch-and-bound code in 2947.32 s on a CDC-CYBER76. Clausen and Perregaard [15] were able to solve instances up to size 20 for the first time in 1994. Their results are published in 1997. Their method, a parallel branch-and-bound algorithm using Gilmore–Lawler lower bound, required 811,440 CPU seconds and traversed 360,148,026 nodes to prove optimality of the incumbent for nug20. Bruengger et al. [19] were the first ones to solve nug21 and nug22 in 1996. In the same year, Clausen et al. [20] reportedly solved nug24. Marzetta and Brüngger managed to solve nug25 in 1999 [21]. Finally, Anstreicher et al. [16] were able to solve nug27, nug28, and nug30 to optimality in the year 2000. The last three instances required the equivalent of 0.18, 0.88, and 6.94 CPU years of time on a HP9000 C3000 workstation, respectively.

This set of instances is not fully representative of the overall computational state of the art for the QAP. As of this writing, the largest instances reportedly solved are ste36a, ste36b, and ste36c that are of size 36. These instances were proposed by Steinberg in 1961 [22]. Brixius and Anstreicher [3] solved ste36a in 2001 with a serial implementation of a branch-and-bound algorithm that uses the Gilmore–Lawler bound, whereas ste36b and ste36c are solved by Nyström [4] in 2001 using distributed computing. Solving ste36a required 180 h on a PIII 800 Mhz PC with a serial implementation, while ste36b and ste36c took approximately 60 days and 200 days of CPU time, respectively. However, instances proposed by Burkard and Offermann in 1977 [23] of size 26 have remained unsolved until recently, at which time they were solved by the method of Hahn et al. [24] (March 2004). There are still instances of size 30 waiting to be solved in the QAPLIB. As a final note, we emphasize the fact that the largest instances solved to date are solved by means of parallel implementations that rely on high amounts of computing power.

The instances rouxx and taixx have flow and distance matrices that are randomly generated, exhibit no discernible patterns, and are the hardest instances in the QAPLIB. For example, solving tai25a required 393.5 days on one 420 MHz CPU of a HPJ5000 workstation by an algorithm using Gilmore–Lawler type of bounds that exploits the binary structure of pairwise assignment matrix [24].

6. Computational results

We have implemented a depth-first branch-and-cut algorithm using IP2′ and the valid inequalities presented in Section 4. Flow diagram of the algorithm is given in Fig. 1.

For instances with symmetric distance matrices, we have made use of the facts $d_{ij} = d_{ji}\ \forall\ i, j$ and $d_{ii} = 0\ \forall i$ that imply $t_{ij} = t_{ji} \forall i, j$, and $t_{ii} = 0\ \forall i$, to decrease the number of t variables by more than one half. Recall that t_{ij} represents the induced distance between facilities i and j. The nonlinear representation of t_{ij} in terms of the assignment variables is $t_{ij} = \sum_{k,l=1}^{n} d_{kl} x_{ik} x_{jl}$. If we were able to completely linearize the objective function, the equality above would hold for all i, j for the solution of the LP relaxation of IP2′. When that is not the case, we can compute the violation of t^*_{ij} as $v^*_{ij} = t^*_{ij} - \sum_{k,l=1}^{n} d_{kl} x^*_{ik} x^*_{jl}$. Notice that v_{ij} approximates the error of linearization, and that the error becomes more severe as $|v_{ij}|$ increases. Similarly, the error of the distance between facilities i and j becomes more important as the amount of flow between the facilities deviates from the average flow. We compute the value $a_{ij} = |v_{ij}| * (|f_{ij} - \text{avg}(\boldsymbol{F})| + \text{avg}(\boldsymbol{F}))$ for all i, j, where avg($\boldsymbol{F}$) denotes the average flow, as an indicator of the importance of linearization. All distances from and to facility i are completely linearized, as soon as facility i is assigned to some location. Consequently, we select as our branching rule an unassigned facility i with the largest sum of $\sum_{j=1}^{n} a_{ij}$.We use *row branching* where child problems are formed by assigning an unassigned facility to all unassigned locations [16].

For instances representing symmetric grid graphs, namely nugxx and scrxx, we have implemented a symmetry test proposed by Mautor and Roucairol [25], which aims to reduce the number of subproblems at each node of the branch-and-cut tree by identifying sets of symmetric locations and branching on a single element from each set. Even though the symmetry test of Mautor and Roucairol is not generally valid for all distance matrices, it is known to be valid for grid graphs with Manhattan metric. This class includes instances nugxx and scrxx. The test proved to be very useful, effectively decreasing the CPU time to one quarter of the original or less.

For computational testing of our algorithm, we used problem instances available from the QAPLIB. We have attempted to solve all sets of problems with the exception of the two sets of data supplied by Eschermann and Wunderlich [26], and Li and Pardalos [27]. The former set of data involves a high level of symmetry in both flow and distance matrices, which renders branching efforts fruitless, and is unsuitable for computational testing. The latter set consists of asymmetric instances with known optimal solutions. This set of problems is not used in the literature for computational testing, hence we leave it out due to lack of a basis of comparison with other methods. In addition to the instances from the QAPLIB, we have created five new instances of sizes 22, 24, 26, 28, and 30, referred to as erd22, erd24, erd26, erd28, and erd30, respectively, in exactly the same way as the instances hadxx are created. We have used GRASP [28] with 10,000 restarts as the startup heuristic. In 89% of the instances, GRASP found the optimum solution. For the cases when the initial incumbent was not optimal, the gap between the optimal solution and the incumbent was at most 2.3%. Application of GRASP did not take more than a few minutes for any of the instances, so we report only the CPU times for the branch-and-cut algorithm. CPLEX 8.11 LP solver was used for optimizing the resulting linear programs. The runs were conducted on a single PC (1133 Mhz Dell PowerEdge with 256 MB RAM). Memory requirement was not more than 15 MB for even the largest instances. The constraint (16) was removed from the formulation and added to the valid inequality pool, to benefit from a smaller ($O(n)$) static constraint set. As empirical proof of the strength of the valid inequalities we have proposed we give, in Table 1 , the number of cuts added at the root node and the effect on the optimum relaxation value. The separated values in columns 2, 3, and 4 are

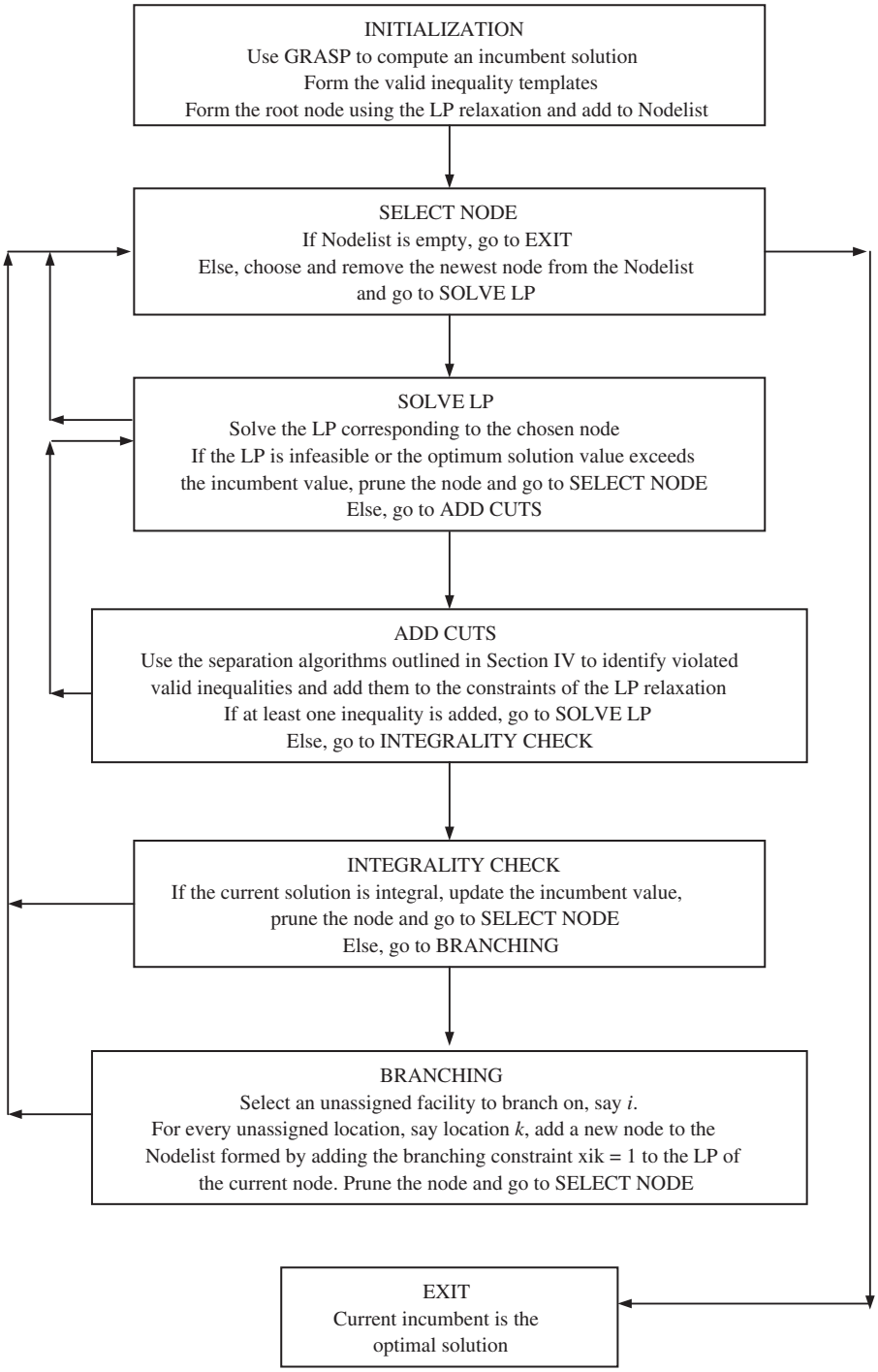

Fig. 1. Flow diagram for the proposed branch-and-cut algorithm.

Table 1
Effect of valid inequalities on lower bound

Data File	Number of cuts added	Initial relaxation value	Final relaxation value	Optimum solution value	Initial gap (%)	Final gap (%)	Data type
bur26a	737	5321819.20	5333986.45	5426670.00	1.93	1.71	General
bur26b	784	3725027.10	3748891.39	3817852.00	2.43	1.81	General
bur26c	741	5320676.00	5333544.83	5426795.00	1.96	1.72	General
bur26d	790	3725693.80	3747069.19	3821225.00	2.50	1.94	General
bur26e	687	5310894.40	5318727.83	5386879.00	1.41	1.27	General
bur26f	774	3713578.20	3732885.17	3782044.00	1.81	1.30	General
bur26g	677	9991175.90	10010200.17	10117172.00	1.25	1.06	General
bur26h	793	6994747.10	7030401.88	7098658.00	1.46	0.96	General
chr12a	53	8448.70	8840.10	9552.00	11.55	7.45	General
chr12b	57	7298.40	8966.50	9742.00	25.08	7.96	General
chr12c	46	9784.40	9909.80	11156.00	12.29	11.17	General
chr15a	85	7607.70	8084.00	9896.00	23.12	18.31	General
chr15b	93	5063.20	6127.20	7990.00	36.63	23.31	General
chr15c	70	8823.00	9129.70	9504.00	7.17	3.94	General
chr18a	104	9014.60	9611.70	11098.00	18.77	13.39	General
chr18b	0	1534.00	1534.00	1534.00	0.00	0.00	General
chr20a	216	2156.00	2156.00	2192.00	1.64	1.64	General
chr20b	134	2236.90	2241.70	2298.00	2.66	2.45	General
chr20c	159	9134.40	12029.50	14142.00	35.41	14.94	General
chr22a	177	5952.90	6021.70	6156.00	3.30	2.18	General
chr22b	159	6018.70	6066.50	6194.00	2.83	2.06	General
chr25a	174	3136.70	3328.10	3796.00	17.37	12.33	General
els19	234	6090771.50	15822114.20	17212548.00	64.61	8.08	General
erd22	369	7780.30	8556.90	8608.00	9.62	0.59	PartialGrid
erd24	429	9486.60	10511.80	10596.00	10.47	0.79	Partial Grid
erd26	439	10859.00	12102.00	12222.00	11.15	0.98	Partial Grid
erd28	486	13661.70	15185.80	15334.00	10.91	0.97	Partial Grid
erd30	554	17188.20	19070.00	19238.00	10.65	0.87	Partial Grid
had12	104	1568.00	1640.30	1652.00	5.08	0.71	Partial Grid
had14	137	2536.40	2710.50	2724.00	6.89	0.50	Partial Grid
had16	193	3392.70	3690.70	3720.00	8.80	0.79	Partial Grid
had18	230	4853.60	5287.00	5358.00	9.41	1.33	Partial Grid
had20	284	6290.20	6852.80	6922.00	9.13	1.00	Partial Grid
nug12	133	457.50	548.70	578.00	20.85	5.07	Grid
nug14	166	815.80	966.90	1014.00	19.55	4.64	Grid
nug15	168	910.10	1099.40	1150.00	20.86	4.40	Grid
nug16a	192	1257.00	1534.90	1610.00	21.93	4.66	Grid
nug16b	164	914.00	1195.00	1240.00	26.29	3.63	Grid
nug17	222	1293.70	1627.70	1732.00	25.31	6.02	Grid
nug18	241	1436.60	1808.20	1930.00	25.56	6.31	Grid

Table 1 (*continued*)

nug20	268	1851.70	2416.70	2570.00	27.95	5.96	Grid
nug21	346	1673.50	2270.90	2438.00	31.36	6.85	Grid
nug22	319	2256.80	3395.20	3596.00	37.24	5.58	Grid
nug24	418	2255.50	3272.90	3488.00	35.34	6.17	Grid
nug25	443	2504.00	3498.30	3744.00	33.12	6.56	Grid
nug27	447	3326.30	4896.34	5236.00	36.47	6.49	Grid
nug28	604	3168.10	4815.96	5166.00	38.67	6.78	Grid
nug30	615	3745.20	5693.97	6124.00	38.84	7.02	Grid
rou12	121	161123.80	211372.80	235528.00	31.59	10.26	General
rou15	211	222087.20	307352.50	354210.00	37.30	13.23	General
scr12	114	26152.00	30793.60	31410.00	16.74	1.96	Grid
scr15	221	39898.20	50222.50	51140.00	21.98	1.79	Grid
scr20	329	75420.00	98995.80	110030.00	31.46	10.03	Grid
tai12a	123	158216.70	207774.30	224416.00	29.50	7.42	General
tai12b	105	11426178.00	36789487.70	39464925.00	71.05	6.78	General
tai15a	205	244239.70	340869.50	388214.00	37.09	12.20	General
tai15b	187	50094649.20	51485572.30	51765268.00	3.23	0.54	General

the objective values corresponding, respectively, to the initial relaxation, final relaxation, and the optimal solution of IP2′.

In some cases, especially when the optimality gap of the initial relaxation is large, the valid inequalities are tremendously effective. For example, for els19 the gap is reduced from 64.61% to 8.08%, and for tai12b, the gap is reduced from 71.05% to 6.78%. In cases when the optimality gap is smaller, varying effects of 0.1–20% is observed. When the distance matrix represents a grid (namely for the instances erdxx, hadxx, nugxx, and scrxx) the effect is stronger. We note that no more than a few hundred inequalities are required even for the largest instances, after removing redundant inequalities.

Details of the branch-and-cut applied to the same problems are given in Table 2 .

We solved all instances of chrxx, including chr25a, in less than 1 h of CPU time. For this set of instances only, we have used IP2 instead of IP2′, since these instances have a special flow matrix that represents a tree, which can be exploited. All instances of hadxx, for which the distance matrix represents a partial grid network (a subgraph of a grid), are solved within 3 min of CPU time. The instance els19, which consists of real world data and exhibits a distance pattern that is quite close to being planar, is solved in about 1 min of CPU time. The instances nugxx, especially nug20 and nug24 proved to be harder. For example, nug20 took about 4 h of CPU time and nug24 took approximately 58 h of CPU time. Notably, computing time requirement decreased for instances nug20, nug21, and nug22 as the size of the problem increased. This is mainly because of the decreasing level of symmetry in 4*5, 3*7, and 2*11 grids. The instances scrxx were solved relatively easily due to the erratic structure of the flow matrix, which is sparse and helps us to branch efficiently. The instances rouxx and taixx, being randomly generated and exhibiting no discernible patterns, are the hardest of all. We solved rou15 in close to 2 h of CPU time and tai15a in about 4.5 h of CPU time, while sizes of $n > 15$ exceed computational reach for these two classes of instances.

Table 2
Problems solved to optimality by branch-and-cut

Data file	Problem size	Number of nodes	CPU time (s)	Initial incumbent value	Optimum solution value	Data type
bur26e	26	59761	81731.28	5386879.00	5386879.00	General
bur26f	26	10439	25154.93	3782044.00	3782044.00	General
bur26g	26	118782	98666.64	10117172.00	10117172.00	General
bur26h	26	4618	10709.26	7098658.00	7098658.00	General
chr12a	12	34	0.96	9552.00	9552.00	General
chr12b	12	12	0.35	9742.00	9742.00	General
chr12c	12	89	2.25	11156.00	11156.00	General
chr15a	15	321	15.98	9896.00	9896.00	General
chr15b	15	165	9.19	7990.00	7990.00	General
chr15c	15	15	1.57	9504.00	9504.00	General
chr18a	18	196	18.91	11098.00	11098.00	General
chr18b	18	0	0.01	1534.00	1534.00	General
chr20a	20	1152	264.32	2192.00	2192.00	General
chr20b	20	1198	259.53	2352.00	2298.00	General
chr20c	20	469	96.77	14142.00	14142.00	General
chr22a	22	2141	394.57	6266.00	6156.00	General
chr22b	22	3340	617.37	6314.00	6194.00	General
chr25a	25	9885	2941.85	4250.00	3796.00	General
els19	19	161	66.29	17212548.00	17212548.00	General
erd22	22	393	199.75	8608.00	8608.00	Partial Grid
erd24	24	3634	1946.51	10596.00	10596.00	Partial Grid
erd26	26	19759	14978.78	12222.00	12222.00	Partial Grid
erd28	28	74923	83504.84	15334.00	15334.00	Partial Grid
erd30	30	90444	127158.06	19238.00	19238.00	Partial Grid
had12	12	12	0.85	1652.00	1652.00	Partial Grid
had14	14	27	1.83	2724.00	2724.00	Partial Grid
had16	16	76	10.06	3720.00	3720.00	Partial Grid
had18	18	717	110.44	5358.00	5358.00	Partial Grid
had20	20	743	156.32	6922.00	6922.00	Partial Grid
nug12	12	43	3.17	578.00	578.00	Grid
nug14	14	747	49.82	1014.00	1014.00	Grid
nug15	15	315	27.45	1150.00	1150.00	Grid
nug16a	16	1856	185.45	1610.00	1610.00	Grid
nug16b	16	334	43.20	1240.00	1240.00	Grid
nug17	17	7652	1020.04	1732.00	1732.00	Grid
nug18	18	20353	3551.20	1930.00	1930.00	Grid
nug20	20	50862	13859.07	2570.00	2570.00	Grid
nug21	21	18537	6019.89	2438.00	2438.00	Grid
nug22	22	10370	3314.30	3596.00	3596.00	Grid
nug24	24	322443	208288.91	3488.00	3488.00	Grid

Table 2 (*continued*)

rou12	12	392	46.98	235528.00	235528.00	General
rou15	15	23469	7248.02	354210.00	354210.00	General
scr12	12	22	1.38	31410.00	31410.00	Grid
scr15	15	23	5.95	51140.00	51140.00	Grid
scr20	20	5161	1467.80	110030.00	110030.00	Grid
tai12a	12	132	15.74	224416.00	224416.00	General
tai12b	12	157	6.82	39464925.00	39464925.00	General
tai15a	15	56406	16490.41	388214.00	388214.00	General
tai15b	15	402	116.65	51765268.00	51765268.00	General

Table 3
Suboptimally solved problems

Data file	Gap depth: 0(%)	Depth: 1(%)	Depth: 2(%)	Depth: 3(%)	CPU time (s)
bur26a	1.71	1.37	1.18	0.91	75711.46
bur26b	1.81	1.49	1.18	0.76	74572.85
bur26c	1.72	1.47	1.26	1.02	132849.90
bur26d	1.94	1.61	1.08	0.75	59443.98
nug25	6.56	6.22	5.56	4.20	16267.40
nug27	6.49	5.85	4.68	3.90	43729.22
nug28	6.78	6.37	5.50	4.42	66432.41
nug30	7.02	6.82	6.15	5.04	130163.14

The instances erdxx were solved quite easily with respect to their size. The largest of them, erd30, took about one and a half days to complete. The computational success mainly depends on the structure of the distance and flow matrices; the former is the shortest distance matrix of a partial grid, and the latter is uniformly drawn from the interval $[1 \ldots n]$. This structure, in turn, yields a strong lower bound and results in a small branch-and-cut tree.

There are also instances, not reported in Table 2, that are attempted but not solved to optimality. For example, branch-and-cut trees of the instances bur26a-d and nug25–30 could not be entirely fathomed. For these problems, we have imposed a depth limit of three and fathomed the reduced trees to collect further data about the strength of the lower bound at the lower nodes of the tree. Instances bur26a-d were not solved despite the fact that instances bur26e-h, which have the same distance matrix but different flow matrices, were solved relatively easily. This suggests that our branching rule performs better for instances bur26e-h, since the branching rule is the only part that depends on the flow matrix. Instances nug25-30 do not yield good lower bounds even in the lower branches due to the high level of symmetry in the distance matrix. These instances simply require more computing power. The data for suboptimally solved problems are summarized in Table 3.

Even though the instances nugxx are considered to be a benchmark, drastic improvements in computing hardware and inherent differences between sequential and parallel implementations increase the difficulty

Table 4
Comparison of scaled CPU times (in minutes) for instances nugxx

Data file	Erdoğan and Tansel	Brixius and Anstreicher
nug16b	0.80	0.90
nug18	66.15	69.20
nug20	258.15	145.80
nug21	112.13	212.30
nug22	61.74	134.30
nug24	3879.89	5829.90

of comparison. For example, nug20 required 811440.0 CPU seconds of a state of the art computer in 1994, when it was solved for the first time by Clausen and Perregaard [15]. Our method requires 13859.07 CPU seconds for the same instance on a PC, but it should be noted that the computing technology has doubled the speed of computation a few times in the past decade. To date, most successful study in terms of dealing with the instances nugxx is that of Anstreicher, Brixius, Goux, and Linderoth [16], which is a parallel branch-and-bound implementation that uses the bound presented in the study by Brixius and Anstreicher [29]. Since our implementation is sequential, we compare our results with the results presented in the latter paper. The authors report solution times for the instances nug16b, nug18, nug20, nug21, nug22, and nug24 as CPU minutes of a HP9000 C3000 workstation, whereas our solution times are for a single PC (1133 Mhz Dell PowerEdge with 256 MB RAM). The CPU's under consideration are different in terms of architecture and speed. For an accurate comparison, we have used the results of a benchmarking study of Guest [30] to scale the run times. Relative to the benchmark system, our system is cited to have 19% CPU performance for floating point operations, whereas HP9000 C3000 is cited to have 17% performance. Hence, we have multiplied our CPU times by 19/17 to have a scaled comparison.

Table 4 gives the comparison of the two studies in terms of CPU time requirements. While Brixius and Anstreicher can solve nug20 with greater efficiency, our method performs better for the larger instances nug21, nug22, and nug24. Note that the CPU time requirement of our method is no more than 65% of the method of Brixius and Anstreicher for these instances. Such benchmarking studies generally produce approximations that are in at most 30% error (Anstreicher [31]). If we allow a liberal error margin of 30% for the results of the benchmarking study, Table 4 gives us reason to claim that our method can compete with the state-of-the-art methods in the literature.

To have a better understanding of the performance of the branch-and-cut algorithm, we have disabled the GRASP heuristic for a few instances that seem to be representative of their corresponding problem sets, and analyzed the change in CPU time and number of nodes traversed. The results are given in Table 5.

As expected, CPU times are longer when the initial upper bound value is set to infinity as compared to the case where the initial upper bound is supplied by the GRASP heuristic. However, characteristics of each instance dictate the order and quality of the integral solutions found by the branch-and-cut algorithm. Hence, the results are somewhat erratic, even among the members of the same instance set. For example, while chr15a suffers a 29% increase in CPU time and 19% increase in the number of nodes traversed, CPU time required by the instance chr20a increases more than four times and the number of nodes traversed increases more than six times when the initial upper bound is set to infinity. In contrast, scr20 performs much better than scr15, resulting in 50% increase in CPU time versus a 179% increase.

Table 5
Run rimes with different initial upper bound values

Data file	ub: infinity		ub:GRASP	
	Number of nodes	CPU time (s)	Number of nodes	CPU time (s)
chr12a	67	1.69	34	0.96
had12	45	1.77	12	0.85
nug12	53	3.39	43	3.17
rou12	1066	99.13	392	46.98
scr12	82	3.34	22	1.38
chr15a	381	20.61	321	15.98
had16	166	19.45	76	10.06
nug15	647	51.41	315	27.45
rou15	35803	10618.13	23469	7248.02
scr15	184	16.59	23	5.95
chr20a	7197	1253.72	1152	264.32
had20	12877	2660.57	743	156.32
nug20	86399	21874.33	50862	13859.07
scr20	8163	2202.93	5161	1467.80

We emphasize the fact that the algorithm we have presented is designed to prove optimality rather than to find good solutions and depends heavily on the quality of the initial solution. In fact, the quality of the initial solutions supplied by the GRASP heuristic encouraged us to use the depth-first node selection rule. A more robust branch-and-cut algorithm may be implemented by switching the node selection rule to breadth-first or best-first.

To better understand the cases in which our algorithm performs best, we have computed the flow dominance and distance dominance of the instances we have attempted to solve. We have also included some metrics (indices) that we have devised. Namely, we have computed the ratio of number of solutions for which the valid inequalities are binding to the total number of solutions. The results are given in Table 6.

The column labeled "Distance Upper Bound Index" denotes the ratio of the number of strict inequalities to the total number of inequalities in the optimum solution of LP1, when solved to optimality with objective function coefficients of $-1/n$. This index value is a measure of the strength of the constructed inequalities for $m = 2$, and can be easily computed by solving a single instance of LP1. Recall that the constructed inequalities for $m = 3$ consist of a single set of coefficients that is applied to all facility triplets. Consider any three facilities and all possible location assignments to these facilities, and count the cases for which the constructed inequalities are strict. The column labeled "Triangle Sum Upper Bound Index" denotes the ratio of this count to the total number of assignments $\left(\binom{n}{3}\right)$. In other words, this index value measures the strength of the constructed valid inequalities for $m=3$. This index can be computed by solving a single linear program with n variables and $\binom{n}{3}$ constraints, and executing the counting process ($O(n^3)$). Finally, the column labeled "Triangle Diff. Upper Bound Index" denotes the ratio of the number of location triplets for which the triangle inequalities are strict to the total number of triplets. Computation of this index requires $O(n^3)$ time. The rest of the columns consist of the indices for the lower bound counterparts

Table 6
Indices computed for the instances

Data file	Flow Dominance	Distance Dominance	Distance Lower Bound Index	Distance Upper Bound Index	Triangle Sum Lower Bound Index	Triangle Sum Upper Bound Index	Triangle Diff. Lower Bound Index	Triangle Diff. Upper Bound Index
bur26a	274.744	15.074	0.197	0.614	0.232	0.074	0.087	0.093
bur26b	274.744	15.901	0.262	0.614	0.302	0.068	0.259	0.093
bur26c	228.227	15.074	0.245	0.614	0.232	0.074	0.087	0.093
bur26d	228.227	15.901	0.226	0.614	0.302	0.068	0.259	0.093
bur26e	253.807	15.074	0.191	0.629	0.232	0.074	0.087	0.093
bur26f	253.807	15.901	0.235	0.726	0.302	0.068	0.259	0.093
bur26g	279.687	15.074	0.215	0.614	0.232	0.074	0.087	0.093
bur26h	279.687	15.901	0.257	0.614	0.302	0.068	0.259	0.093
chr12a	63.206	307.980	0.182	0.833	0.564	0.155	0.071	0.018
chr12b	63.206	307.980	0.485	0.833	0.605	0.055	0.092	0.018
chr12c	63.206	307.980	0.250	0.833	0.545	0.064	0.042	0.018
chr15a	69.735	326.951	0.262	0.867	0.635	0.053	0.051	0.011
chr15b	69.735	326.951	0.433	0.867	0.651	0.095	0.073	0.012
chr15c	69.735	326.951	0.143	0.867	0.629	0.037	0.027	0.012
chr18a	63.098	350.595	0.248	0.889	0.692	0.088	0.046	0.007
chr18b	56.863	356.319	0.176	0.889	0.686	0.032	0.019	0.007
chr20a	59.385	345.940	0.197	0.900	0.723	0.040	0.038	0.006
chr20b	59.385	345.940	0.124	0.900	0.716	0.036	0.015	0.006
chr20c	65.630	345.940	0.447	0.900	0.736	0.335	0.059	0.006
chr22a	66.887	420.620	0.273	0.909	0.747	0.057	0.045	0.005
chr22b	66.887	420.620	0.199	0.909	0.742	0.034	0.024	0.005
chr25a	57.925	423.928	0.267	0.920	0.776	0.098	0.041	0.006
els19	530.281	52.030	0.164	0.129	0.022	0.029	0.007	0.014
erd22	45.955	64.209	0.541	0.216	0.038	0.362	0.009	0.267
erd24	46.502	63.699	0.554	0.217	0.033	0.313	0.017	0.270
erd26	45.756	61.711	0.502	0.209	0.030	0.291	0.010	0.253
erd28	44.751	62.042	0.545	0.196	0.021	0.290	0.015	0.254
erd30	44.053	63.933	0.547	0.182	0.023	0.284	0.008	0.260
had12	50.679	63.130	0.682	0.364	0.109	0.418	0.018	0.333
had14	49.456	66.622	0.560	0.297	0.071	0.451	0.013	0.328
had16	48.403	64.829	0.542	0.292	0.050	0.454	0.014	0.300
had18	47.132	63.681	0.542	0.255	0.048	0.397	0.012	0.272
had20	45.957	64.243	0.553	0.221	0.042	0.385	0.006	0.277
nug12	116.580	56.891	0.576	0.424	0.155	0.382	0.055	0.255
nug14	103.566	56.749	0.582	0.374	0.113	0.352	0.043	0.244
nug15	106.476	56.582	0.562	0.362	0.101	0.255	0.040	0.245
nug16a	100.737	57.334	0.542	0.325	0.086	0.325	0.020	0.257
nug16b	115.595	54.772	0.500	0.342	0.093	0.371	0.019	0.238
nug17	104.827	56.259	0.522	0.324	0.078	0.344	0.021	0.236
nug18	104.211	54.935	0.529	0.301	0.072	0.348	0.019	0.229

Table 6 (*continued*)

nug20	103.646	54.102	0.489	0.284	0.061	0.309	0.014	0.228
nug21	117.061	57.385	0.514	0.267	0.053	0.264	0.018	0.235
nug22	114.216	64.086	0.459	0.216	0.038	0.291	0.010	0.262
nug24	112.783	54.130	0.457	0.243	0.043	0.261	0.008	0.221
nug25	110.763	53.033	0.480	0.240	0.041	0.257	0.010	0.217
nug27	111.402	58.614	0.487	0.211	0.032	0.194	0.010	0.229
nug28	112.999	54.499	0.450	0.212	0.032	0.260	0.007	0.217
nug30	112.417	52.725	0.448	0.202	0.029	0.227	0.006	0.210
rou12	67.053	71.538	0.182	0.182	0.055	0.055	0.018	0.018
rou15	68.739	69.073	0.148	0.143	0.033	0.035	0.011	0.011
rou20	65.569	64.352	0.105	0.105	0.018	0.018	0.006	0.006
scr12	256.487	56.891	0.576	0.424	0.155	0.382	0.055	0.255
scr15	247.750	54.921	0.533	0.371	0.103	0.404	0.040	0.236
scr20	254.333	54.102	0.489	0.284	0.061	0.309	0.014	0.228
tai12a	74.765	69.307	0.182	0.174	0.055	0.055	0.021	0.018
tai12b	299.606	79.211	0.212	0.189	0.055	0.082	0.024	0.045
tai15a	70.563	63.777	0.152	0.143	0.035	0.033	0.012	0.013
tai15b	312.935	262.313	0.324	0.176	0.035	0.068	0.015	0.092

of the same valid inequalities. Note that, a higher value means a stronger effect, but the values across the columns are not comparable, since the corresponding valid inequalities differ in strength.

Notice that for the instances chrxx, the values of Distance Upper Bound Index and Triangle Sum Lower Bound Index are very high, justifying the ease of solution for these instances. For the instances erdxx and hadxx, the values of the indices for Distance Lower Bound, Triangle Sum Upper Bound, and Triangle Difference Upper Bound are notably high. Although the same indices are remarkably high for the instances nugxx and scrxx, the symmetry factor comes into play and increases the level of difficulty. Instances burxx exhibit high values for Distance Upper Bound Index and Triangle Sum Lower Bound Index. Consequently, the lower bounds generated at the root node are close to the optimum solution value. Instances rouxx and taixx do not exhibit any high values for any of the indices, and hence, are the hardest of all.

The flow and distance dominance values do not mean much without the rest of the data. For example, the instances chrxx exhibit large distance dominance values, the instances scrxx exhibit large flow dominance values, and finally the instances erdxx exhibit low values for both and distance dominance. All three sets of instances have been solved with reasonable efficiency, so the dominance values seem irrelevant. However, further analysis of the instances scr12, scr15, and scr20 and nug12, nug15, and nug20 that have the same distance matrices reveals that the scrxx instances are solved with greater efficiency. The only apparent reason for this is the higher flow dominance value of these instances. Likewise the instance els19, which does not yield high values for any of the indices but has the highest flow dominance value, is solved efficiently. We find it appropriate to conclude that a higher value of flow dominance helps our branching strategy to find the decisions that are more important than the others.

We can conclude that our method performs best when one or more of the following occur:

(1) A value of 0.5 or higher for at least one of Distance Upper Bound and Distance Lower Bound Indices.
(2) A value of 0.3 or higher for at least one of Triangle Sum Upper Bound and Triangle Sum Lower Bound Indices.
(3) A value of 0.3 or higher for at least one of Triangle Diff. Upper Bound and Triangle Diff. Lower Bound Indices.
(4) A flow dominance value of 200 or more.

7. Conclusion

We developed a new perspective of modeling the QAP based on a flow interpretation of the problem. To date, decisions about the flows induced between locations (or the distances induced between facilities) were ignored, because they were dominated by the assignment decisions. This dominance is to be expected, as the flows (distances) are immediately determined when the assignment decisions are made. However, by incorporating these subdecisions explicitly into our models, we are able to exploit any underlying structure available in the data. The particular feature that we focused on is the triangle inequalities in the distance matrix. The formulations offer insights and further motivate us to question the paradigms of modeling up to now.

In operations research, the common approach is to model the problem on hand in a way that is as independent from the problem instance as possible. Modelers often tend to dump the data into the objective function and focus on solving a well-defined, static constraint matrix. In the case of the QAP, this paradigm has been unfruitful beyond certain sizes, due to the large number of variables required. In our opinion, to be able to solve the QAP exactly, one needs to incorporate the data into the model, and uncover the pattern beneath the data. In the case where one of the matrices is a distance matrix, which obeys triangle inequalities, the pattern is apparent. In cases with randomly created data matrices, the problem becomes much harder to solve.

An unforeseen consequence of incorporating the data into the constraint matrix is the need to solve auxiliary problems in order to identify valid inequalities. To be more precise, for the TSP, for example, one can logically identify the subtour elimination constraints. However, in our case, we need to solve LPs to identify or to *construct* valid inequalities. Thus, we can claim that with a constraint matrix dependent on the problem instance at hand, the act of building more information into the model becomes a problem of its own.

We have tried to analyze the behavior of the algorithm we have presented using different metrics we have devised, as well as metrics from the literature. We have observed that our algorithm performs best when one or more of the proposed metrics is significantly high. Using the formulations and valid inequalities we have presented, we have been able to solve an instance of size 30 that exhibits high values for the metrics we have presented. In contrast, we have failed to solve instances from the randomly created sets of problems that are of size larger than 15.

We have focused on identifying all violated valid inequalities, for the sake of a better analysis. It is our belief that with high-performance heuristics to identify violated valid inequalities, and access to higher amounts of computing power, the proposed models may be used to solve larger problem instances.

References

[1] Sahni S, Gonzalez T. P-complete approximation problems. Journal of the Association of Computing Machinery 1976;23:555–65.
[2] Anstreicher KM, Brixius NW. A New Bound for the quadratic assignment problem based on convex quadratic programming. Technical report, Department of Management Sciences, University of Iowa, 1999.
[3] Brixius NW, and Anstreicher KM. The Steinberg wiring problem. Working Paper, The University of Iowa, 2001.
[4] Nyström M. Solving certain large instances of the quadratic assignment problem: steinberg's examples. Working paper, California Institute of Technology, 1999.
[5] Applegate D, Bixby R, Chvatal V, Cook W, Helsgaun K. Traveling Salesman Problem Homepage. url: http://www.tsp.gatech.edu/, 2004.
[6] Koopmans TC, Beckmann M. Assignment problems and the location of economic activities. Econometrica 1957;25: 53–76.
[7] Lawler E. The quadratic assignment problem. Management Science 1963;9:586–99.
[8] Adams WP, Johnson TA. Improved linear programming bounds for the quadratic sssignment problem. In: Pardalos PM, Wolkowicz H, editors Quadratic Assignment and Related Problems. DIMACS Series on Discrete Mathematics and Theoretical Computer Science 16, 1994, AMS, Providence, RI. p. 43–75.
[9] Finke G, Burkard RE, Rendl F. Quadratic assignment problems. Annals of Discrete Mathematics 1987;31:61–82.
[10] Kaufmann L, Broeckx F. An algorithm for the quadratic assignment problem. European Journal of Operational Research 1978;2:204–11.
[11] Ramakrishnan KG, Resende MGC, Pardalos PM. A Branch-and-bound algorithm for the quadratic assignment problem using a lower bound based on linear programming. In: Floudas C, Pardalos PM, editors. State of the art in global optimization. Dordrecht: Kluwer Academic Publishers; 1995.
[12] Kettani O, Oral M. Reformulating quadratic assignment problems for efficient optimization. IIE Transactions 1993;25: 97–107.
[13] Burkard RE, Karisch SE, Rendl F. QAPLIB—a quadratic assignment problem library. Journal of Global Optimization 1997;10:391–403.
[14] Nugent CE, Vollman TE, Ruml J. An experimental comparison of techniques for the assignment of facilities to locations. Operations Research 1968;16:150–73.
[15] Clausen J, Perregaard M. Solving large quadratic assignment problems in parallel. Computational Optimization and Applications 1997;8:11–127.
[16] Anstreicher KM, Brixius NW, Goux J-P, Linderoth J. Solving large quadratic assignment problems on computational grids. Mathematical Programming 2002;91:563–88.
[17] Burkard R, Stratmann K. Numerical investigations on quadratic assignment problems. Naval Research Logistics Quarterly 1978;25:129–48.
[18] Burkard RE, Derigs U. Assignment and matching problems. European Journal of Operational Research 1983;13:374–86.
[19] Bruengger A, Clausen J, Marzetta A, Perregaard M. Joining forces in solving large-scale quadratic assignment problems in parallel. DIKU Technical report, University of Copenhagen, 1996.
[20] Clausen J, Espersen T, Karisch SE, Perregaard M, Sensen N, Tschöke S. Benchmark testing for quadratic assignment problems on a portable parallel branch-and-bound library. Work in progress, 1996.
[21] Marzetta A, and Brungger A. A dynamic programming bound for the quadratic assignment problem. In: Computing and combinatorics: fifth annual international conference COCOON'99, LNCS, vol. 1627, Heidelberg: Springer, 1999, p. 339–48.
[22] Steinberg L. The backboard wiring problem: a placement algorithm. SIAM Review 1961;3:37–50.
[23] Burkard R, Offermann J. Entwurf von Schreibmaschinentastaturen mittels quadratischer Zuordnungsprobleme. Zeitschrift für Operations Research 1977;21:B121–32.
[24] Hahn P, Hightower WL, Johnson TA, Guignard-Spielberg M, Roucairol C. Tree elaboration strategies in branch and bound algorithms for solving the quadratic assignment problem. Yugoslavian Journal of Operational Research 2001;11:41–60.
[25] Mautor T, Roucairol C. A new exact algorithm for the solution of quadratic assignment problems. Discrete Applied Mathematics 1994;55:281–93.
[26] Eschermann B, Wunderlich HJ. Optimized synthesis of self-testable finite state machines In: twentieth international symposium on fault-tolerant computing (FFTCS 20), Newcastle upon Tyne, 26–28th June, 1990.

[27] Li Y, Pardalos PM. Generating quadratic assignment test problems with known optimal permutations. Computational Optimization and Applications 1992;1:163–84.
[28] Li Y, Pardalos PM, Resende MGC. A greedy randomized adaptive search procedure for the quadratic assignment problem. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 1994;16:237–61.
[29] Brixius NW, Anstreicher KM. Solving quadratic assignment problems using convex quadratic programming relaxations. Optimization Methods and Software 2001;16:49–68.
[30] Guest MF. GAMESS-UK Benchmarks. url: http://www.cfs.dl.ac.uk/benchmarks/gamess_uk.html, 2005.
[31] Anstreicher KM, Personal communication.